FFTs on BG/L Machines

Vienna University of Technology
Institute for Applied Mathematics
and Numerical Analysis

SFB AURORA

Franz Franchetti

Topics

- Self-adapting DSP software
  - State of the art DSP software
    - SPIRAL
    - FFTW
  - Why is adaptivity needed?
- How to adapt FFTs to BG/L automatically?
  - Utilizing Hummer^2
  - Utilizing the communication network

BG/L Hardware Characteristics

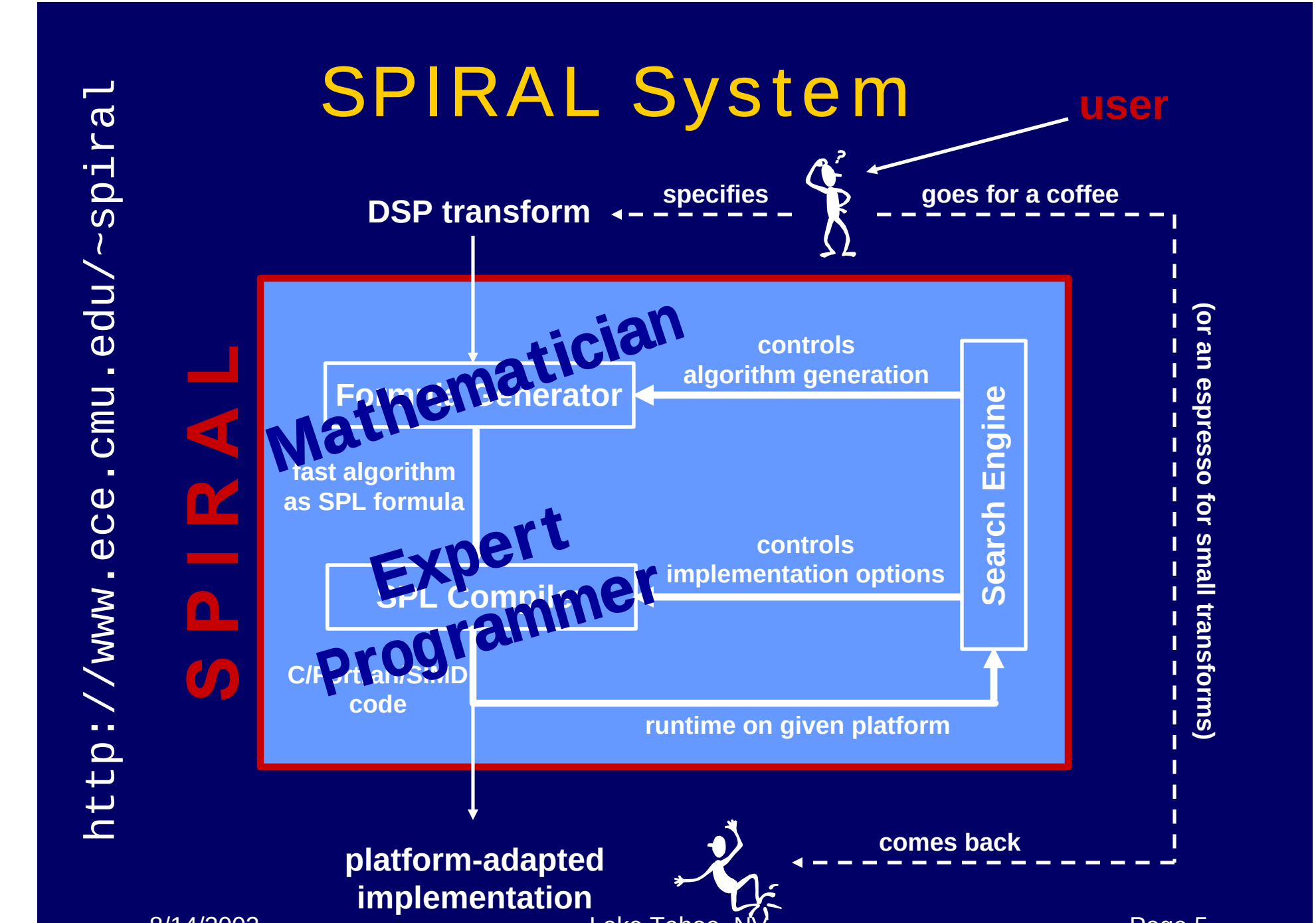
- CPU extensions : **vectorization**
  - Hummer^2 two-way SIMD
- Two-processor nodes: **overlapping techniques**
  - Computation processor
  - Communication processor
- Communication Network: **optimal schedule**
  - 3D torus
  - Fill along lines

FFT Algorithms: Theory vs. Implementation

- Discrete Fourier transform: $O(n^2)$ operations
- Fast Fourier transform: $O(n \log n)$ operations

- Different algorithm families
  - Cooley-Tukey
  - Good-Thomas
  - Rader
  - Bluestein
- Vast number of possible algorithms
- Algorithms mainly differ in memory access patterns
- Runtime varies tremendously

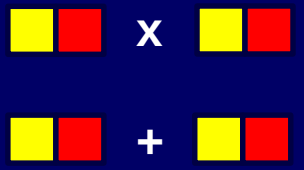
SPIRAL System



SIMD Short Vector Extensions

- Extension to instruction set architecture
- Available on most current architectures (SSE/SSE2 on Intel, AltiVec on Motorola G4, 3DNow! On AMD, Hummer^2 on BG/L)
- Originally for multimedia (like MMX for integers)
- Requires fine grain parallelism
- Large potential speed-up

(2-way)
vector length = 2



Problems:

- SIMD instructions are architecture specific
- No common API (usually assembly hand coding)
- Performance very sensitive to memory access
- Automatic vectorization (by compilers) very limited

very difficult to use

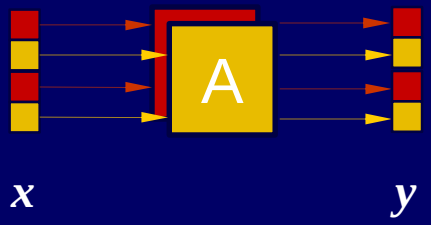
Vectorization in SPIRAL

(Franz Franchetti and Markus Püschel)

Naturally vectorizable construct

$$A \otimes I_2$$

vector length



(Current) generic construct completely vectorizable:

$$\prod_{i=1}^{k} P_i D_i (A_i \otimes I_u) E_i Q_i$$

P_i, Q_i	**permutations**
D_i, E_i	**diagonals**
A_i	**arbitrary formulas**
?	**SIMD vector length**

Vectorization in two steps:

1. **Formula manipulation using manipulation rules**
2. **Code generation (vector code + C code)**

FFTW – The Fastest Fourier Transform in the West

(Matteo Frigo and Steven Johnson)

- Adaptive Library for FFTs
  - Various basic routines combined to compute the whole transform
  - Combination determined at runtime by dynamic programming
  - Basic routines generated automatically
- Main Components
  - Codelets
  - Planner
  - Executor
  - Generator
- Features
  - 1D and MD transforms
  - Arbitrary vector lengths and strides
  - Serial and parallel implementations

`http://www.fftw.org`

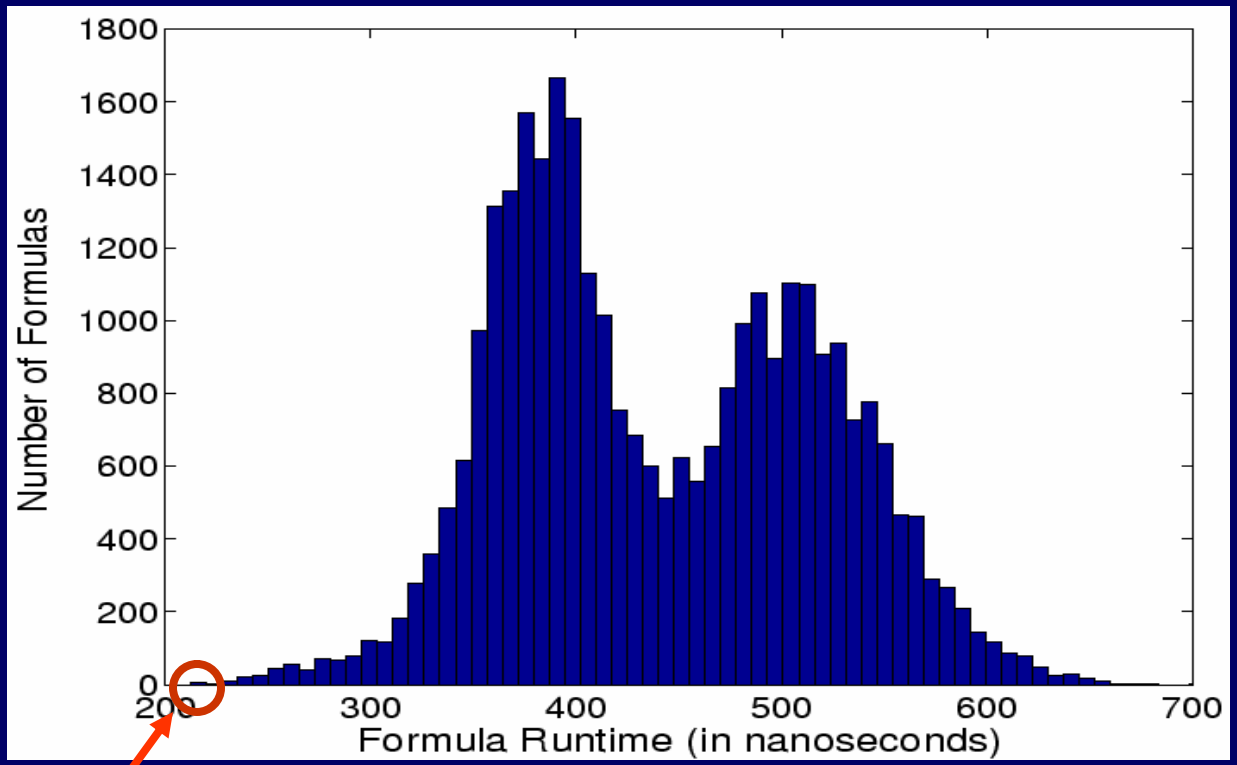
Automatic Vectorization in FFTW

(Stefan Kral)

- 2-way vectorizer for straight line code
  - Rule based
- Special purpose assembly backend
  - Register allocation
  - Index computation
- Integrated into the FFTW codelet generator
- Available to download for
  - AMD K6-2
  - AMD K7
  - Intel Pentium 4

`http://www.fftw.org/~skral`

Performance Study: DCT



DCT, type IV, size 16

~31000 formulas

Best formula

- **All automatically implemented**
- **Large spread in runtimes, even for modest size**
- **Not due to arithmetic cost**
- **Best formula is platform-dependent**

Performance Study: FFT

(1998)

FFT Program	Vector Lengths			
	2^5	2^{10}	2^{15}	2^{20}
NAG/`c60fcf`	11.6	6.0	3.3	2.6
IMSL/`dfftcf`	2.0	1.7	2.7	3.9
Numerical Recipies/`four1`	2.6	2.1	2.2	3.9
FFTPACK/`cfftf`	1.4	**1.0**	2.1	4.0
Green	1.6	1.1	**1.0**	—
FFTW	**1.0**	1.1	1.1	**1.0**

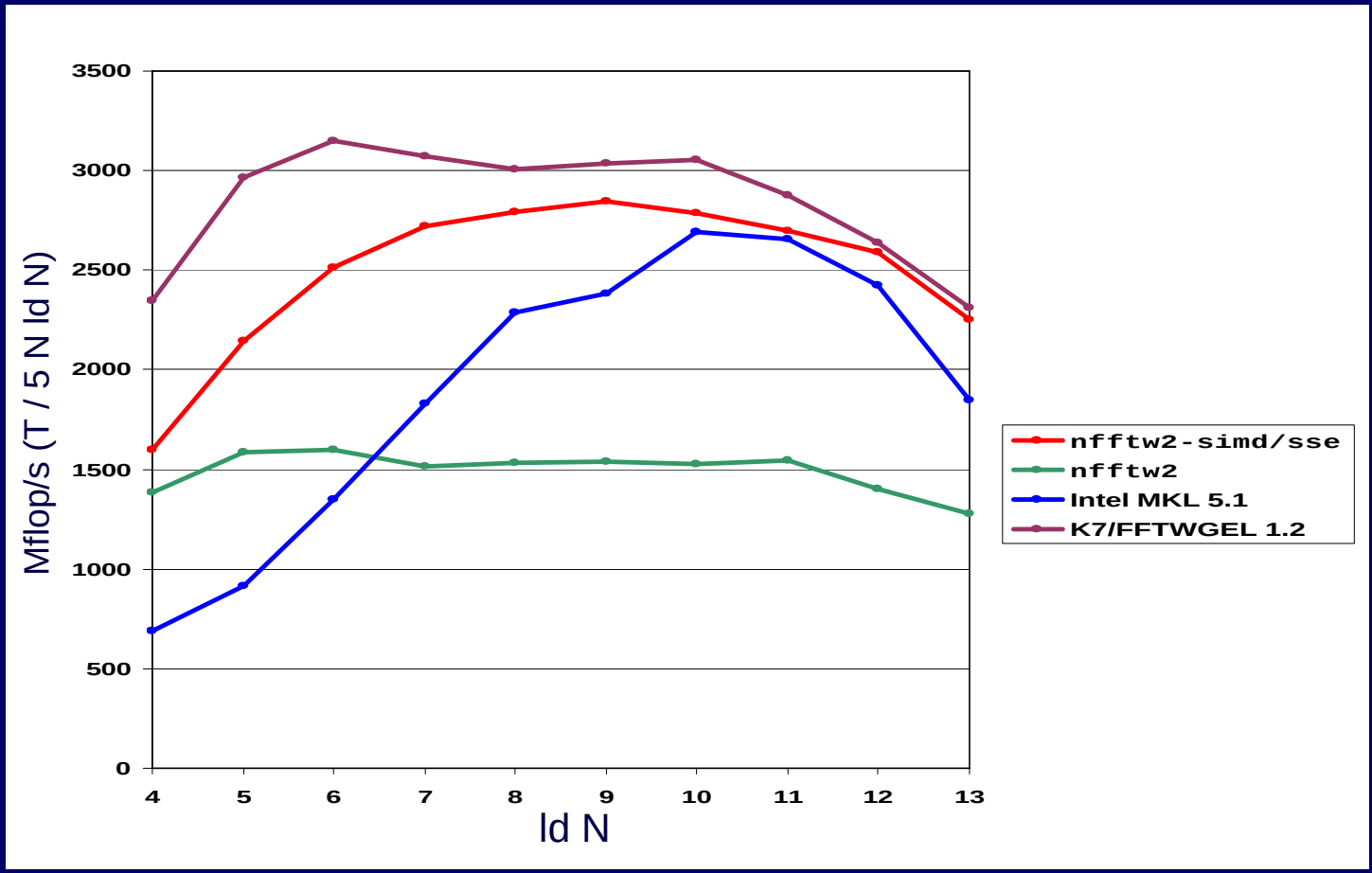
Slow-down factors

One processor of an SGI Power Challenge XL

Adaptive software outperforms commercial software (NAG, IMSL) significantly

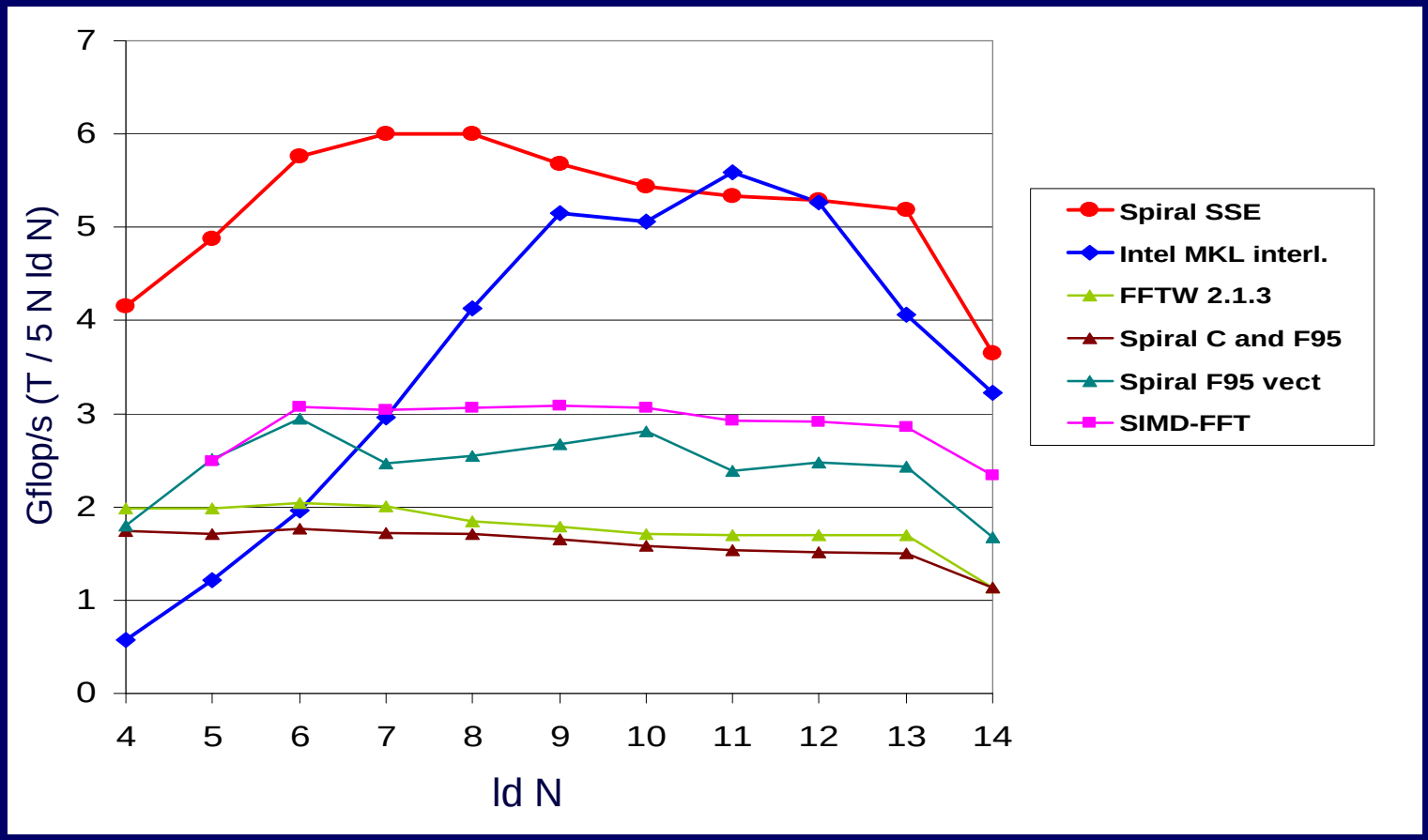
SIMD Vectorization: FFTW

AMD AthlonXP 1800+ (1533 MHz), *Single precision*,
Intel C++ Compiler 5.0, gcc 2.96 and g77 2.96

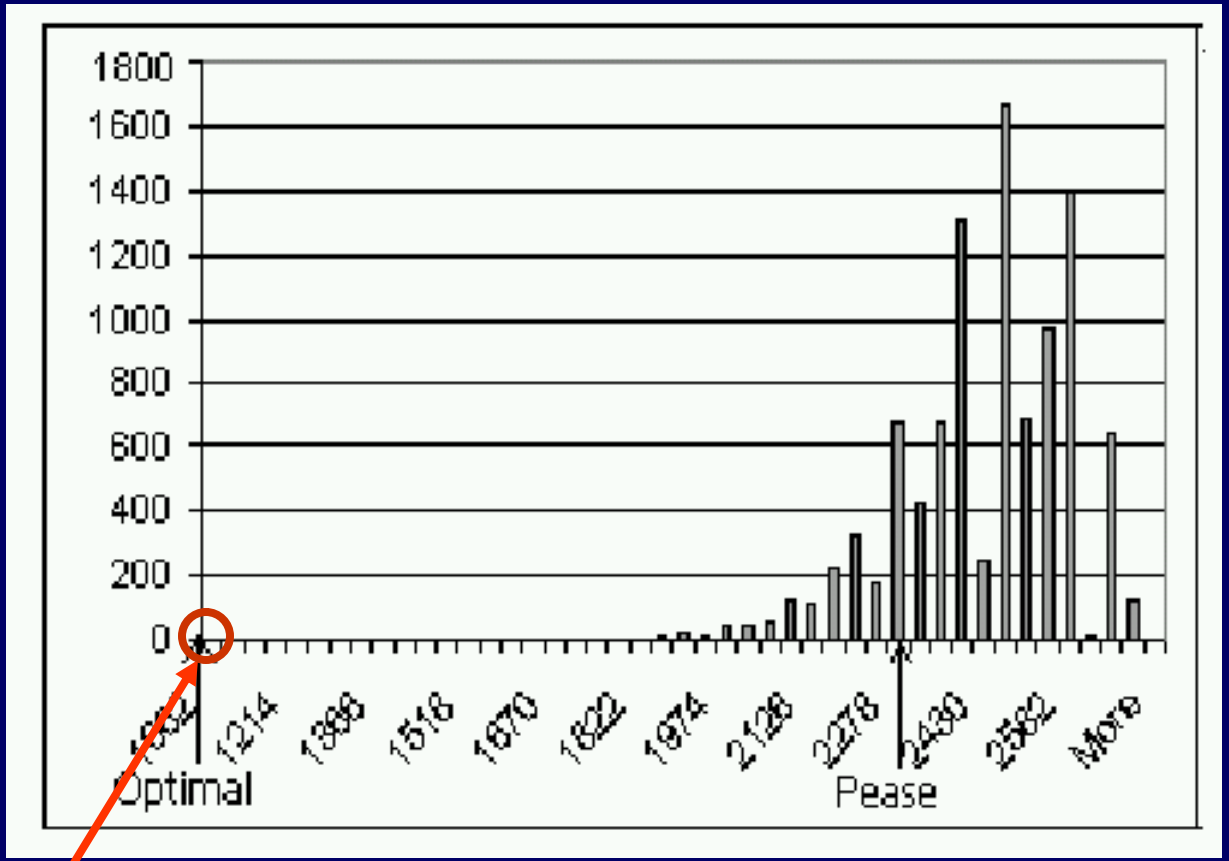


SIMD Vectorization: SPIRAL

Intel Pentium 4 (2530 MHz), S*ingle precision,* Intel C++ Compiler 6.0



Communication



Performance distribution

10000 data flows

Best data flow

- Problem: Find optimal communication network for FFTs
- Evaluate given networks w.r.t data flows that occur in FFTs

Self-adapting FFTs for BG/L

Algorithm decomposed into

- **Adaptive communication layer**
  - Find optimal communication pattern on BG/L for given *FFT problem* and *nodes*
- **Adaptive computation layer**
  - Find optimal computing kernels w.r.t. communication pattern
  - utilize Hummer^2

FFT kernels generated **automatically**

Dimensionless FFT

(Jeremy Johnson et al.)

- DFT of Abelian groups
  - Unifies 1D and MD DFTs, Cooley-Tukey, Good-Thomas, Vector Radix FFT,...
- Automatic optimization of parallel FFT algorithms
  - Data flow description of FFTs based on *permutations*
  - Adaptation of parallel FFT algorithms to architecture
- Find optimal data flow by
  - Simulation
  - Symbolic analysis

Overlapping Algorithm

(Herbert Karner)

- Parallel 1D FFT algorithm
- Communication and computation overlapped
- Based on *2-level six-step* recursion
- Utilizes any local FFT kernel
- Parallel 1D kernel for MD FFTs

Reduced Transform Algorithm

(Tolimieri et al.)

- Multidimensional algorithm for non-powers of 2
- 3 phases
  - *Overlapped* download + summation
  - Local FFTs (*No communication!*)
  - Upload
- Lower count of 1D FFTs
- Based on multiplicative structure of index sets

Approximative FFT

(Alan Edelman)

- SVD based
  - Only significant singular values used
  - Involves special matrix-vector products
- Saves communication
- Increases arithmetic complexity
  - Big DFT ? smaller matrix-vector products
- Decreases accuracy

FFTs for BG/L

- Single processor efforts
  - SIMD vectorization
    - Automatic 2-way vectorization
    - Portabel *n*-way vectorization
  - FMA optimization
- Parallel machine efforts
  - Dimensionless FFT
  - Overlapping
  - Adaptive communication
  - Reduced Transform Algorithm

Conclusion

FFT algorithms for BG/L

- **Automatic *code gegeration* and *adaptation***
  - Standard algorithms
  - Vectorization
  - Parallelization

  is required
- Related results prove performance